Exhibit C-2

storage, the method including maintenance of plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	checkpoints, a copy-on-write snapshot of the file system. VxFS storage checkpoints can be mounted as writable checkpoints using the –o rw or –o remount options. (*See* p. 86, p. 89.) When such a storage checkpoint is mounted as writable, it initially shares data with the original file system through copy-on-write techniques. For instance, at pp. 84-85, VxFS teaches that "[t]he Storage Checkpoint presents the exact image of the file system by finding the data from the primary fileset. . . . This is called the *copy-on-write* technique, which allows the Storage Checkpoint to preserve the image of the primary fileset when the Storage Checkpoint is taken." Once a storage checkpoint is mounted as writable, it is inherent that it becomes an independent active file system. For instance, VxFS at p. 86 states that "[y]ou can mount, access, and write to a data Storage Checkpoint just as you would to a file system." Furthermore, VxFS at p. 89 warns that such a writable checkpoint will diverge from a read-only storage checkpoint when it is written to. Changes made to a writable storage checkpoint are not reflected in the original file system.
2. A method as in claim 1, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.	VxFS at pp. 83-85 and elsewhere discloses that storage checkpoints are made using copy-on-write techniques. When such a storage checkpoint is mounted as writable, it initially shares data with the original file system.
3. A method as in claim 2, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Because VxFS storage checkpoints use copy-on-write techniques, as disclosed at pp. 83-85 and elsewhere, it is inherent that changes to either the writable storage checkpoint or the original file system are stored in different locations so as to not overwrite any live data.
4. A method as in claim 2, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Because VxFS storage checkpoints use copy-on-write techniques, as disclosed at pp. 83-85 and elsewhere, it is inherent that changes to either the writable storage checkpoint or the original file system are stored in different locations so as to not overwrite any live data.
5. A method as in claim 1,	VxFS at p. 83 teaches that storage checkpoints can be taken

wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.	of multiple file systems. Thus, at p. 83 it is disclosed that [y]ou can create a Storage Checkpoint on a single file system or a list of file systems. A multiple file system Storage Checkpoint simultaneously freezes the file systems, creates a Storage Checkpoint on all file systems, and thaws the file systems." As disclosed at pp. 82-85, each checkpoint is an image of its file system at a past consistency point. Thus, at p. 83 it is disclosed that "[t]he storage checkpoint is logically identical to the primary fileset when the storage checkpoint is created." At pp. 84-85, VxFS teaches that "[t]he Storage Checkpoint presents the exact image of the file system by finding the data from the primary fileset. . . . This is called the *copy-on-write* technique, which allows the Storage Checkpoint to preserve the image of the primary fileset when the Storage Checkpoint is taken."
6. A method as in claim 5, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	VxFS discloses on p. 86 and elsewhere that each storage snapshot contains file system metadata. For instance, as disclosed at pp. 83-85, this metadata includes pointers to data blocks. Appendix C further discloses that VxFS metadata includes a superblock, inodes and pointer tables for each file system.
7. A method as in claim 5, wherein at least one of the snapshots is converted into a new active file system.	VxFS discloses at pp. 83-85 and 89 that checkpoints can be taken of an active file system and that storage checkpoints can be mounted as writable, becoming other active file systems.
8. A method as in claim 7, wherein the one of the snapshots is converted by making the one of the snapshots writable.	VxFS discloses at pp. 83-85 and 89 that checkpoints can be taken of an active file system and that storage checkpoints can be mounted as writable, becoming other active file systems.
9. A method as in claim 8, wherein snapshot pointers from any of the active file systems to the new active file system are severed.	Once a storage checkpoint is mounted as writable, it is inherent that it becomes an independent active file system. For instance, VxFS at p. 86 states that "[y]ou can mount, access, and write to a data Storage Checkpoint just as you would to a file system." Furthermore, VxFS at p. 89 warns that such a writable checkpoint will diverge from a read-only storage checkpoint when it is written to. Therefore, no pointer exists between active file systems in VxFS.
10. A method of creating plural active file systems, comprising the steps of: making a snapshot of a	VxFS discloses a file system. Chapter 8 discloses storage checkpoints, a copy-on-write snapshot of the file system. VxFS storage checkpoints can be mounted as writable

first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	checkpoints using the –o rw or –o remount options. (*See* p. 86, p. 89.) When such a storage checkpoint is mounted as writable, it initially shares data with the original file system through copy-on-write techniques. For instance, at pp. 84-85, the VxFS teaches that "[t]he Storage Checkpoint presents the exact image of the file system by finding the data from the primary fileset. . . . This is called the *copy-on-write* technique, which allows the Storage Checkpoint to preserve the image of the primary fileset when the Storage Checkpoint is taken." Once a storage checkpoint is mounted as writable, it is inherent that it becomes an independent active file system. For instance, VxFS at p. 86 states that "[y]ou can mount, access, and write to a data Storage Checkpoint just as you would to a file system." Furthermore, VxFS at p. 89 warns that such a writable checkpoint will diverge from a read-only storage checkpoint when it is written to. Thus, changes made to a writable storage checkpoint are not reflected in the original file system.
11. A method as in claim 10, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Because VxFS storage checkpoints use copy-on-write techniques, as disclosed at pp. 83-85 and elsewhere, it is inherent that changes to either the writable storage checkpoint or the original file system are stored in different locations so as to not overwrite any live data.
12. A method as in claim 10, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Because VxFS storage checkpoints use copy-on-write techniques, as disclosed at pp. 83-85 and elsewhere, it is inherent that changes to either the writable storage checkpoint or the original file system are stored in different locations so as to not overwrite any live data.
13. A method as in claim 10, further comprising the step of severing any snapshot pointers from the first active file system to the second active file system.	Once a storage checkpoint is mounted as writable, it is inherent that it becomes an independent active file system. For instance, VxFS at p. 86 states that "[y]ou can mount, access, and write to a data Storage Checkpoint just as you would to a file system." Furthermore, VxFS at p. 89 warns that such a writable checkpoint will diverge from a read-only storage checkpoint when it is written to. Therefore, no pointer exists between active file systems in VxFS.
14. A method as in claim 10,	VxFS at p. 83 teaches that storage checkpoints can be taken

further comprising the steps of making snapshots of ones of the plural active file systems.	of multiple file systems. Thus, at p. 83 it is disclosed that [y]ou can create a Storage Checkpoint on a single file system or a list of file systems. A multiple file system Storage Checkpoint simultaneously freezes the file systems, creates a Storage Checkpoint on all file systems, and thaws the file systems." At pp. 84-85, VxFS teaches that "[t]he Storage Checkpoint presents the exact image of the file system by finding the data from the primary fileset. . . . This is called the *copy-on-write* technique, which allows the Storage Checkpoint to preserve the image of the primary fileset when the Storage Checkpoint is taken."
15. A method as in claim 14, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	VxFS discloses on p. 86 and elsewhere that each storage snapshot contains file system metadata. For instance, as disclosed at pp. 83-85, this metadata includes pointers to data blocks. Appendix C further discloses that VxFS metadata includes a superblock, inodes and pointer tables for each file system.
16. A method as in claim 10, further comprising the steps of: making anew snapshot of the first active file system, the new snapshot initially sharing data with the first active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	VxFS storage checkpoints can be mounted as writable checkpoints using the –o rw or –o remount options. (*See* p. 86, p. 89.) When such a storage checkpoint is mounted as writable, it initially shares data with the original file system through copy-on-write techniques. Once a storage checkpoint is mounted as writable, it is inherent that it becomes an independent active file system. For instance, VxFS at p. 86 states that "[y]ou can mount, access, and write to a data Storage Checkpoint just as you would to a file system." Furthermore, VxFS at p. 89 warns that such a writable checkpoint will diverge from a read-only storage checkpoint when it is written to. Thus, changes made to a writable storage checkpoint are not reflected in the original file system. Once a VxFS storage checkpoint is mounted as an active file system, a storage snapshot can be taken of it.
17. A method as in claim 16, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Because VxFS storage checkpoints use copy-on-write techniques, as disclosed at pp. 83-85 and elsewhere, it is inherent that changes to any newly mounted active file systems and changes to the "parent" file system are stored in different locations so as to not overwrite any live data.

18. A method as in claim 10, further comprising the steps of: making a new snapshot of the second active file system, the new snapshot initially sharing data with the second active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	VxFS storage checkpoints can be mounted as writable checkpoints using the –o rw or –o remount options. (*See* p. 86, p. 89.) When such a storage checkpoint is mounted as writable, it initially shares data with the original file system through copy-on-write techniques. Once a storage checkpoint is mounted as writable, it is inherent that it becomes an independent active file system. For instance, VxFS at p. 86 states that "[y]ou can mount, access, and write to a data Storage Checkpoint just as you would to a file system." Furthermore, VxFS at p. 89 warns that such a writable checkpoint will diverge from a read-only storage checkpoint when it is written to. Thus, changes made to a writable storage checkpoint are not reflected in the original file system. Once a VxFS storage checkpoint is mounted as an active file system, a storage snapshot can be taken of it.
19. A method as in claim 18, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Because VxFS storage checkpoints use copy-on-write techniques, as disclosed at pp. 83-85 and elsewhere, it is inherent that changes to any newly mounted active file systems and changes to the "parent" file system are stored in different locations so as to not overwrite any live data.
20. A memory storing information including instructions, the instructions executable by a processor to operate data storage, the instructions comprising steps to maintain plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	VxFS discloses a file system. It is inherent that the file system is comprised of computer-executable instructions. Chapter 8 discloses storage checkpoints, a copy-on-write snapshot of the file system. VxFS storage checkpoints can be mounted as writable checkpoints using the –o rw or –o remount options. (*See* p. 86, p. 89.) When such a storage checkpoint is mounted as writable, it initially shares data with the original file system through copy-on-write techniques. For instance, at pp. 84-85, VxFS teaches that "[t]he Storage Checkpoint presents the exact image of the file system by finding the data from the primary fileset. . . . This is called the *copy-on-write* technique, which allows the Storage Checkpoint to preserve the image of the primary fileset when the Storage Checkpoint is taken." Once a storage checkpoint is mounted as writable, it is inherent that it becomes an independent active file system. For instance, VxFS at p. 86 states that "[y]ou can mount, access, and write to a data Storage Checkpoint just as you would to a file system." Furthermore, VxFS at p. 89 warns

	that such a writable checkpoint will diverge from a read-only storage checkpoint when it is written to. Thus, changes made to a writable storage checkpoint are not reflected in the original file system.
21. A memory as in claim 20, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.	VxFS at pp. 83-85 and elsewhere discloses that storage checkpoints are made using copy-on-write techniques. When such a storage checkpoint is mounted as writable, it initially shares data with the original file system.
22. A memory as in claim 21, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Because VxFS storage checkpoints use copy-on-write techniques, as disclosed at pp. 83-85 and elsewhere, it is inherent that changes to either the writable storage checkpoint or the original file system are stored in different locations so as to not overwrite any live data.
23. A memory as in claim 21, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Because VxFS storage checkpoints use copy-on-write techniques, as disclosed at pp. 83-85 and elsewhere, it is inherent that changes to either the writable storage checkpoint or the original file system are stored in different locations so as to not overwrite any live data.
24. A memory as in claim 20, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.	VxFS at p. 83 teaches that storage checkpoints can be taken of multiple file systems. Thus, at p. 83 it is disclosed that [y]ou can create a Storage Checkpoint on a single file system or a list of file systems. A multiple file system Storage Checkpoint simultaneously freezes the file systems, creates a Storage Checkpoint on all file systems, and thaws the file systems." As disclosed at pp. 82-85, each checkpoint is an image of its file system at a past consistency point. Thus, at p. 83 it is disclosed that "[t]he storage checkpoint is logically identical to the primary fileset when the storage checkpoint is created." At pp. 84-85, VxFS teaches that "[t]he Storage Checkpoint presents the exact image of the file system by finding the data from the primary fileset. . . . This is called the *copy-on-write* technique, which allows the Storage Checkpoint to preserve the image of the primary fileset when the Storage Checkpoint is taken."

25. A method as in claim 24, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	VxFS discloses on p. 86 and elsewhere that each storage snapshot contains file system metadata. For instance, as disclosed at pp. 83-85, this metadata includes pointers to data blocks. Appendix C further discloses that VxFS metadata includes a superblock, inodes and pointer tables for each file system.
26. A memory as in claim 24, wherein at least one of the snapshots is converted into a new active file system.	VxFS discloses at pp. 83-85 and 89 that checkpoints can be taken of an active file system and that storage checkpoints can be mounted as writable, becoming other active file systems.
27. A memory as in claim 26, wherein the one of the snapshots is converted by making the one of the snapshots writable.	VxFS discloses at pp. 83-85 and 89 that checkpoints can be taken of an active file system and that storage checkpoints can be mounted as writable, becoming other active file systems.
28. A memory as in claim 27, wherein snapshot pointers from any of the active file systems to the new active file system are severed.	Once a storage checkpoint is mounted as writable, it is inherent that it becomes an independent active file system. For instance, VxFS at p. 86 states that "[y]ou can mount, access, and write to a data Storage Checkpoint just as you would to a file system." Furthermore, VxFS at p. 89 warns that such a writable checkpoint will diverge from a read-only storage checkpoint when it is written to. Therefore, no pointer exists between active file systems in VxFS.
29. A memory storing information including instructions, the instructions executable by a processor to create plural active file systems, the instructions comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	VxFS discloses a file system. It is inherent that the file system is comprised of computer-executable instructions. Chapter 8 discloses storage checkpoints, a copy-on-write snapshot of the file system. VxFS storage checkpoints can be mounted as writable checkpoints using the –o rw or –o remount options. (*See* p. 86, p. 89.) When such a storage checkpoint is mounted as writable, it initially shares data with the original file system through copy-on-write techniques. For instance, at pp. 84-85, VxFS teaches that "[t]he Storage Checkpoint presents the exact image of the file system by finding the data from the primary fileset. . . . This is called the *copy-on-write* technique, which allows the Storage Checkpoint to preserve the image of the primary fileset when the Storage Checkpoint is taken." Once a storage checkpoint is mounted as writable, it is inherent that it becomes an independent active file system. For instance, VxFS at p. 86 states that "[y]ou can mount, access, and write to a data Storage Checkpoint just as you would to a file system." Furthermore, VxFS at p. 89 warns that such a writable checkpoint will diverge from a read-

	only storage checkpoint when it is written to. Thus, changes made to a writable storage checkpoint are not reflected in the original file system.
30. A memory as in claim 29, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Because VxFS storage checkpoints use copy-on-write techniques, as disclosed at pp. 83-85 and elsewhere, it is inherent that changes to either the writable storage checkpoint or the original file system are stored in different locations so as to not overwrite any live data.
31. A memory as in claim 29, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Because VxFS storage checkpoints use copy-on-write techniques, as disclosed at pp. 83-85 and elsewhere, it is inherent that changes to either the writable storage checkpoint or the original file system are stored in different locations so as to not overwrite any live data.
32. A memory as in claim 29, wherein the instructions further comprise the step of severing any snapshot pointers from the first active file system to the second active file system.	Once a storage checkpoint is mounted as writable, it is inherent that it becomes an independent active file system. For instance, VxFS at p. 86 states that "[y]ou can mount, access, and write to a data Storage Checkpoint just as you would to a file system." Furthermore, VxFS at p. 89 warns that such a writable checkpoint will diverge from a read-only storage checkpoint when it is written to. Therefore, no pointer exists between active file systems in VxFS.
33. A memory as in claim 29, wherein the instructions further comprise the steps of making snapshots of ones of the plural active file systems.	VxFS at p. 83 teaches that storage checkpoints can be taken of multiple file systems. Thus, at p. 83 it is disclosed that [y]ou can create a Storage Checkpoint on a single file system or a list of file systems. A multiple file system Storage Checkpoint simultaneously freezes the file systems, creates a Storage Checkpoint on all file systems, and thaws the file systems." At pp. 84-85, VxFS teaches that "[t]he Storage Checkpoint presents the exact image of the file system by finding the data from the primary fileset. . . . This is called the *copy-on-write* technique, which allows the Storage Checkpoint to preserve the image of the primary fileset when the Storage Checkpoint is taken."
34. A memory as in claim 33, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from	VxFS discloses on p. 86 and elsewhere that each storage snapshot contains file system metadata. For instance, as disclosed at pp. 83-85, this metadata includes pointers to data blocks. Appendix C further discloses that VxFS

active file system data for the plural active file systems.	metadata includes a superblock, inodes and pointer tables for each file system. The VxFS metadata is unique to each snapshot and to each active file system.
35. A memory as in claim 29, wherein the instructions further comprise the steps of: making a new snapshot of the first active file system, the new snapshot initially sharing data with the first active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	VxFS storage checkpoints can be mounted as writable checkpoints using the –o rw or –o remount options. (*See* p. 86, p. 89.) When such a storage checkpoint is mounted as writable, it initially shares data with the original file system through copy-on-write techniques. Once a storage checkpoint is mounted as writable, it is inherent that it becomes an independent active file system. For instance, VxFS at p. 86 states that "[y]ou can mount, access, and write to a data Storage Checkpoint just as you would to a file system." Furthermore, VxFS at p. 89 warns that such a writable checkpoint will diverge from a read-only storage checkpoint when it is written to. Changes made to a writable storage checkpoint are not reflected in the original file system. Once a VxFS storage checkpoint is mounted as an active file system, a storage snapshot can be taken of it.
36. A memory as in claim 35, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Because VxFS storage checkpoints use copy-on-write techniques, as disclosed at pp. 83-85 and elsewhere, it is inherent that changes to any newly mounted active file systems and changes to the "parent" file system are stored in different locations so as to not overwrite any live data.
37. A memory as in claim 29, wherein the instructions further comprise the steps of: making a new snapshot of the second active file system, the new snapshot initially sharing data with the second active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	VxFS storage checkpoints can be mounted as writable checkpoints using the –o rw or –o remount options. (*See* p. 86, p. 89.) When such a storage checkpoint is mounted as writable, it initially shares data with the original file system through copy-on-write techniques. Once a storage checkpoint is mounted as writable, it is inherent that it becomes an independent active file system. For instance, VxFS at p. 86 states that "[y]ou can mount, access, and write to a data Storage Checkpoint just as you would to a file system." Furthermore, VxFS at p. 89 warns that such a writable checkpoint will diverge from a read-only storage checkpoint when it is written to. Thus, changes made to a writable storage checkpoint are not reflected in the original file system.

	Once a VxFS storage checkpoint is mounted as an active file system, a storage snapshot can be taken of it.
38. A memory as in claim 37, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Because VxFS storage checkpoints use copy-on-write techniques, as disclosed at pp. 83-85 and elsewhere, it is inherent that changes to any newly mounted active file systems and changes to the "parent" file system are stored in different locations so as to not overwrite any live data.
39. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to maintain plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	VxFS discloses a file system. It is inherent that the file system is comprised of computer-executable instructions operable to maintain file systems and acting though a storage controller. It is also well known and understood in the art that data inherently can be sent and received over a network when operating under file system control, such as VxFS. Chapter 8 discloses storage checkpoints, a copy-on-write snapshot of the file system. VxFS storage checkpoints can be mounted as writable checkpoints using the –o rw or –o remount options. (*See* p. 86, p. 89.) When such a storage checkpoint is mounted as writable, it initially shares data with the original file system through copy-on-write techniques. For instance, at pp. 84-85, VxFS teaches that "[t]he Storage Checkpoint presents the exact image of the file system by finding the data from the primary fileset. . . . This is called the *copy-on-write* technique, which allows the Storage Checkpoint to preserve the image of the primary fileset when the Storage Checkpoint is taken." Once a storage checkpoint is mounted as writable, it is inherent that it becomes an independent active file system. For instance, VxFS at p. 86 states that "[y]ou can mount, access, and write to a data Storage Checkpoint just as you would to a file system." Furthermore, VxFS at p. 89 warns that such a writable checkpoint will diverge from a read-only storage checkpoint when it is written to. Changes made to a writable storage checkpoint are not reflected in the original file system.
40. A storage system as in claim 39, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share	VxFS at pp. 83-85 and elsewhere discloses that storage checkpoints are made using copy-on-write techniques. When such a storage checkpoint is mounted as writable, it initially shares data with the original file system.

data.	
41. A storage system as in claim 40, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Because VxFS storage checkpoints use copy-on-write techniques, as disclosed at pp. 83-85 and elsewhere, it is inherent that changes to either the writable storage checkpoint or the original file system are stored in different locations so as to not overwrite any live data.
42. A storage system as in claim 40, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Because VxFS storage checkpoints use copy-on-write techniques, as disclosed at pp. 83-85 and elsewhere, it is inherent that changes to either the writable storage checkpoint or the original file system are stored in different locations so as to not overwrite any live data.
43. A storage system as in claim 39, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.	VxFS at p. 83 teaches that storage checkpoints can be taken of multiple file systems. Thus, at p. 83 it is disclosed that [y]ou can create a Storage Checkpoint on a single file system or a list of file systems. A multiple file system Storage Checkpoint simultaneously freezes the file systems, creates a Storage Checkpoint on all file systems, and thaws the file systems." As disclosed at pp. 82-85, each checkpoint is an image of its file system at a past consistency point. Thus, at p. 83 it is disclosed that "[t]he storage checkpoint is logically identical to the primary fileset when the storage checkpoint is created." At pp. 84-85, VxFS teaches that "[t]he Storage Checkpoint presents the exact image of the file system by finding the data from the primary fileset. . . . This is called the *copy-on-write* technique, which allows the Storage Checkpoint to preserve the image of the primary fileset when the Storage Checkpoint is taken."
44. A storage system as in claim 43, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	VxFS discloses on p. 86 and elsewhere that each storage snapshot contains file system metadata. For instance, as disclosed at pp. 83-85, this metadata includes pointers to data blocks. Appendix C further discloses that VxFS metadata includes a superblock, inodes and pointer tables for each file system.
45. A storage system as in claim 43, wherein at least one of the snapshots is converted into a new	VxFS discloses at pp. 83-85 and 89 that checkpoints can be taken of an active file system and that storage checkpoints can be mounted as writable, becoming other active file

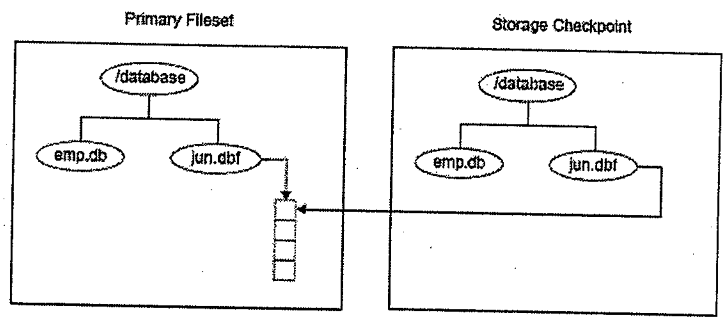
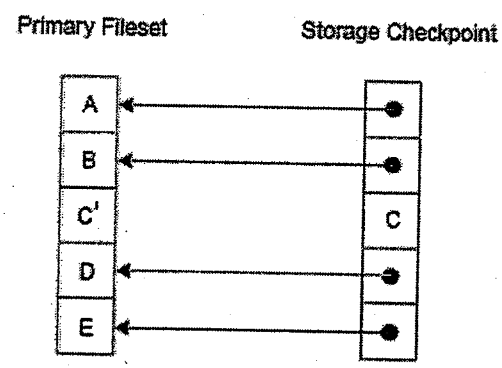
active file system.	systems.
46. A storage system as in claim 45, wherein the one of the snapshots is converted by making the one of the snapshots writable.	VxFS discloses at pp. 83-85 and 89 that checkpoints can be taken of an active file system and that storage checkpoints can be mounted as writable, becoming other active file systems.
47. A storage system as in claim 46, wherein snapshot pointers from any of the active file systems to the new active file system are severed.	Once a storage checkpoint is mounted as writable, it is inherent that it becomes an independent active file system. For instance, VxFS at p. 86 states that "[y]ou can mount, access, and write to a data Storage Checkpoint just as you would to a file system." Furthermore, VxFS at p. 89 warns that such a writable checkpoint will diverge from a read-only storage checkpoint when it is written to. Therefore, no pointer exists between active file systems in VxFS.
48. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to create plural active file systems, the program control comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	VxFS discloses a file system. It is inherent that the file system is comprised of computer-executable instructions operable to maintain file systems and acting though a storage controller. It is also well known and understood in the art that data inherently can be sent and received over a network when operating under file system control, such as VxFS. Chapter 8 discloses storage checkpoints, a copy-on-write snapshot of the file system. VxFS storage checkpoints can be mounted as writable checkpoints using the –o rw or –o remount options. (*See* p. 86, p. 89.) When such a storage checkpoint is mounted as writable, it initially shares data with the original file system through copy-on-write techniques. For instance, at pp. 84-85, VxFS teaches that "[t]he Storage Checkpoint presents the exact image of the file system by finding the data from the primary fileset. . . . This is called the *copy-on-write* technique, which allows the Storage Checkpoint to preserve the image of the primary fileset when the Storage Checkpoint is taken." Once a storage checkpoint is mounted as writable, it is inherent that it becomes an independent active file system. For instance, VxFS at p. 86 states that "[y]ou can mount, access, and write to a data Storage Checkpoint just as you would to a file system." Furthermore, VxFS at p. 89 warns that such a writable checkpoint will diverge from a read-only storage checkpoint when it is written to. Therefore, changes made to a writable storage checkpoint are not reflected in the original file system.
49. A storage system as in claim 48, wherein when changes are	Because VxFS storage checkpoints use copy-on-write techniques, as disclosed at pp. 83-85 and elsewhere, it is

made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	inherent that changes to either the writable storage checkpoint or the original file system are stored in different locations so as to not overwrite any live data.
50. A storage system as in claim 48, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Because VxFS storage checkpoints use copy-on-write techniques, as disclosed at pp. 83-85 and elsewhere, it is inherent that changes to either the writable storage checkpoint or the original file system are stored in different locations so as to not overwrite any live data.
51. A storage system as in claim 48, wherein the program control further comprises the step of severing any snapshot pointers from the first active file system to the second active file system.	Once a storage checkpoint is mounted as writable, it is inherent that it becomes an independent active file system. For instance, VxFS at p. 86 states that "[y]ou can mount, access, and write to a data Storage Checkpoint just as you would to a file system." Furthermore, VxFS at p. 89 warns that such a writable checkpoint will diverge from a read-only storage checkpoint when it is written to. Therefore, no pointer exists between active file systems in VxFS.
52. A storage system as in claim 48, wherein the program control further comprises the steps of making snapshots of ones of the plural active file systems.	VxFS at p. 83 teaches that storage checkpoints can be taken of multiple file systems. Thus, at p. 83 it is disclosed that [y]ou can create a Storage Checkpoint on a single file system or a list of file systems. A multiple file system Storage Checkpoint simultaneously freezes the file systems, creates a Storage Checkpoint on all file systems, and thaws the file systems." At pp. 84-85, VxFS teaches that "[t]he Storage Checkpoint presents the exact image of the file system by finding the data from the primary fileset. . . . This is called the *copy-on-write* technique, which allows the Storage Checkpoint to preserve the image of the primary fileset when the Storage Checkpoint is taken."
53. A storage system as in claim 52, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	VxFS discloses on p. 86 and elsewhere that each storage snapshot contains file system metadata. For instance, as disclosed at pp. 83-85, this metadata includes pointers to data blocks. Appendix C further discloses that VxFS metadata includes a superblock, inodes and pointer tables for each file system. The VxFS metadata is unique to each snapshot and to each active file system.
54. A storage system as in claim 48, wherein the program control	VxFS storage checkpoints can be mounted as writable checkpoints using the –o rw or –o remount options. (*See* p.

further comprises the steps of: making a new snapshot of the first active file system, the new snapshot initially sharing data with the first active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	86, p. 89.) When such a storage checkpoint is mounted as writable, it initially shares data with the original file system through copy-on-write techniques. Once a storage checkpoint is mounted as writable, it is inherent that it becomes an independent active file system. For instance, VxFS at p. 86 states that "[y]ou can mount, access, and write to a data Storage Checkpoint just as you would to a file system." Furthermore, VxFS at p. 89 warns that such a writable checkpoint will diverge from a read-only storage checkpoint when it is written to. Thus, changes made to a writable storage checkpoint are not reflected in the original file system. Once a VxFS storage checkpoint is mounted as an active file system, a storage snapshot can be taken of it.
55. A storage system as in claim 54, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Because VxFS storage checkpoints use copy-on-write techniques, as disclosed at pp. 83-85 and elsewhere, it is inherent that changes to any newly mounted active file systems and changes to the "parent" file system are stored in different locations so as to not overwrite any live data.
56. A storage system as in claim 48, wherein the program control further comprises the steps of: making a new snapshot of the second active file system, the new snapshot initially sharing data with the second active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	VxFS storage checkpoints can be mounted as writable checkpoints using the –o rw or –o remount options. (*See* p. 86, p. 89.) When such a storage checkpoint is mounted as writable, it initially shares data with the original file system through copy-on-write techniques. Once a storage checkpoint is mounted as writable, it is inherent that it becomes an independent active file system. For instance, VxFS at p. 86 states that "[y]ou can mount, access, and write to a data Storage Checkpoint just as you would to a file system." Furthermore, VxFS at p. 89 warns that such a writable checkpoint will diverge from a read-only storage checkpoint when it is written to. Changes made to a writable storage checkpoint are not reflected in the original file system. Once a VxFS storage checkpoint is mounted as an active file system, a storage snapshot can be taken of it.
57. A storage system as in claim 56, wherein when changes are	Because VxFS storage checkpoints use copy-on-write techniques, as disclosed at pp. 83-85 and elsewhere, it is

made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	inherent that changes to any newly mounted active file systems and changes to the "parent" file system are stored in different locations so as to not overwrite any live data.
58. An apparatus for operating data storage, the apparatus including means for creating plural active file systems and means for maintaining plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	VxFS discloses a file system. Chapter 8 discloses storage checkpoints, a copy-on-write snapshot of the file system. VxFS storage checkpoints can be mounted as writable checkpoints using the –o rw or –o remount options. (*See* p. 86, p. 89.) When such a storage checkpoint is mounted as writable, it initially shares data with the original file system through copy-on-write techniques. For instance, at pp. 84-85, VxFS teaches that "[t]he Storage Checkpoint presents the exact image of the file system by finding the data from the primary fileset. . . . This is called the *copy-on-write* technique, which allows the Storage Checkpoint to preserve the image of the primary fileset when the Storage Checkpoint is taken." Once a storage checkpoint is mounted as writable, it is inherent that it becomes an independent active file system. For instance, VxFS at p. 86 states that "[y]ou can mount, access, and write to a data Storage Checkpoint just as you would to a file system." Furthermore, VxFS at p. 89 warns that such a writable checkpoint will diverge from a read-only storage checkpoint when it is written to. Changes made to a writable storage checkpoint are not reflected in the original file system.
59. An apparatus for creating plural active file systems, comprising: means for making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and means for converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	VxFS discloses a file system. Chapter 8 discloses storage checkpoints, a copy-on-write snapshot of the file system. VxFS storage checkpoints can be mounted as writable checkpoints using the –o rw or –o remount options. (*See* p. 86, p. 89.) When such a storage checkpoint is mounted as writable, it initially shares data with the original file system through copy-on-write techniques. For instance, at pp. 84-85, VxFS teaches that "[t]he Storage Checkpoint presents the exact image of the file system by finding the data from the primary fileset. . . . This is called the *copy-on-write* technique, which allows the Storage Checkpoint to preserve the image of the primary fileset when the Storage Checkpoint is taken." Once a storage checkpoint is mounted as writable, it is inherent that it becomes an independent active file system. For instance, VxFS at p. 86 states that "[y]ou can mount, access, and write to a data Storage Checkpoint just as you

Attorney Docket No. 347155-29

	would to a file system." Furthermore, VxFS at p. 89 warns that such a writable checkpoint will diverge from a read-only storage checkpoint when it is written to. Changes made to a writable storage checkpoint are not reflected in the original file system.
60. A method of operating data storage, comprising: making a snapshot of organizational data of a first active file system, the snapshot pointing to original non-organizational data of the first active file system; storing the snapshot; modifying a first portion of the original non-organizational data of the first active file system in response to a first active file system access request, resulting in a modified first portion being part of first modified non-organizational data of the first active file system; and storing the modified first portion so as not to overwrite the first portion; wherein, after the step of storing the modified first portion, the snapshot points to the original non-organizational data, the organizational data of the first active file system point to the first modified non-organizational data of the first filing system, and the original non-organizational data and the first modified non-organizational data partially overlap.	VxFS discloses at pp. 83-85 making a snapshot of organizational data (file system metadata) of a first file system, the snapshot pointing to original non-organizational data (data blocks): Figure 2. Primary Fileset and its Storage Checkpoint  VxFS further teaches at pp. 83-85 that as data blocks are modified, they are stored using copy-on-write so as not to overwrite the old data, which is copied to a new location. After data blocks are updated, the checkpoint points to the original data, which partially overlaps with the changed data blocks: Figure 4. Updates to the Primary Fileset  VxFS further discloses at pp. 83-85 that the active file system uses a block map to point to updated data blocks.
61. A method according to claim 60, wherein: the step of storing the snapshot comprises storing the	VxFS storage checkpoints can be mounted as writable checkpoints using the –o rw or –o remount options. (*See* p. 86, p. 89.) When such a storage checkpoint is mounted as

snapshot as a second active filing system; the method further comprising: modifying a second portion of the original non-organizational data in response to a second active file system access request, resulting in a modified second portion being part of second modified non-organizational data of the second active file system; and storing the modified second portion so as not to overwrite the second portion; wherein, after the step of storing the modified second portion, the snapshot points to the second modified non-organizational data, the organizational data of the first active file system point to the first modified non-organizational data, and the first modified non-organizational data and the second modified non-organizational data partially overlap.	writable, it initially shares data with the original file system through copy-on-write techniques, so as not to overwrite any data. For instance, at pp. 84-85, VxFS teaches that "[t]he Storage Checkpoint presents the exact image of the file system by finding the data from the primary fileset. . . . This is called the *copy-on-write* technique, which allows the Storage Checkpoint to preserve the image of the primary fileset when the Storage Checkpoint is taken." It is inherent that if a VxFS storage checkpoint is mounted as an active file system, its metadata will point to the modified data blocks that are written to it, as well as to unmodified data blocks that it still shares with the parent active file system
62. A method according to claim 61, wherein the step of making a snapshot is performed at a consistency point of the first active file system.	As disclosed at pp. 82-85, each checkpoint is an image of its file system at a past consistency point. Thus, at p. 83 it is disclosed that "[t]he storage checkpoint is logically identical to the primary fileset when the storage checkpoint is created." At pp. 84-85, VxFS teaches that "[t]he Storage Checkpoint presents the exact image of the file system by finding the data from the primary fileset. . . . This is called the *copy-on-write* technique, which allows the Storage Checkpoint to preserve the image of the primary fileset when the Storage Checkpoint is taken."
63. A method according to claim 62, wherein data is stored in the first and second active file systems using blocks.	VxFS at e.g. pp. 83-85 discloses that data is stored using data blocks.

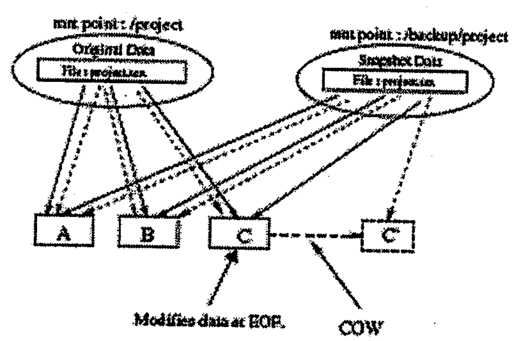
<u>Third Basis of Invalidity</u>

The reference applicable to the third basis of invalidity is:

1. S. B. Siddha, K. Gopinath, *A Persistent Snapshot Device Driver for Linux*, Proceedings of the 5th Annual Linux Showcase & Conference, USENIX, Nov. 5-10, 2001. ("Siddha").

A copy of substantially the same disclosure can also be found in S. B. Siddha, *Persistent Snapshots*, A Project Report Submitted in partial fulfillment of the requirement for the Degree of Master of Engineering in Computer Science and Engineering, INDIAN INSTITUTE OF SCIENCE, January, 2000. ("Siddha Report"). Siddha Report has substantially the same disclosure as the Siddha reference described above, but is dated January 2000. Disclosure of writable snapshots can be found, for example, in section 2.1.3. Therefore, Siddha Report anticipates claims of the '001 patent at least for the same reasons as Siddha.

The pertinence and manner of applying Siddha to claims 1-63 for which re-examination is requested is as follows:

1. A method of operating data storage, the method including maintenance of plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	Siddha generally and section 3.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:  Figure 1: Snapshot COW Mechanism Siddha section 3.1.3 further teaches that snapshots can be made writable, establishing a diverging snapshot tree of multiple active file systems, as depicted in Fig. 4:

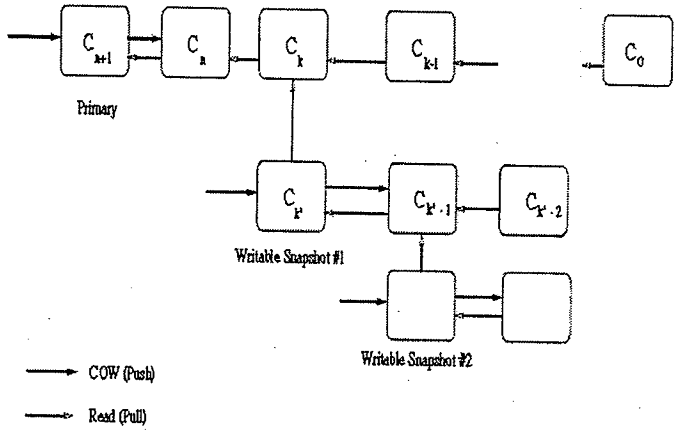
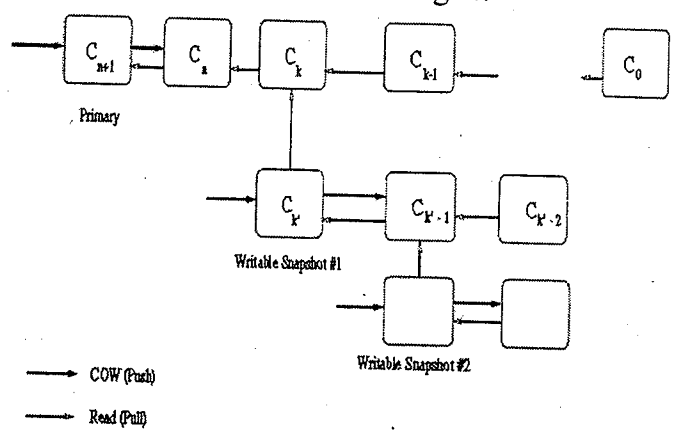
<table>
<tr><td></td><td>

Figure 4: Snapshot Tree
Snapshot Ck' is even labeled as "Writable Snapshot #1." Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 4, writable snapshot Ck' initially shares data with snapshot Ck, which represents the state of the original file system at time k. Snapshot Ck'-1, representing the state of the new file system at a later time, does not share changed data with snapshot Ck-1, which represents the state of the original file system.</td></tr>
<tr><td></td><td></td></tr>
<tr><td>2. A method as in claim 1, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.</td><td>Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 4, writable snapshot Ck' initially shares data with snapshot Ck, which represents the state of the original file system at time k. Snapshot Ck'-1, representing the state of the new file system at a later time, does not share changed data with snapshot Ck-1, which represents the state of the original file system.</td></tr>
<tr><td></td><td></td></tr>
<tr><td>3. A method as in claim 2, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.</td><td>Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.</td></tr>
<tr><td></td><td></td></tr>
<tr><td>4. A method as in claim 2, wherein when changes are</td><td>Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written</td></tr>
</table>

Attorney Docket No. 347155-29

made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	in new locations so as to not overwrite other data, as shown in Fig. 1.
5. A method as in claim 1, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.	Siddha in section 3.1.3 and Fig. 4 teaches snapshots of plural active file systems. Section 3.1.3 further teaches the need to keep snapshots consistent through atomic updates. Fig. 4 shows each snapshot to be a past consistent view of active file systems. Section 3.1.3 further teaches that "[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4."  Figure 4: Snapshot Tree
6. A method as in claim 5, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	Sections 1.1, 3.1.3 and Figs. 1 and 4 teach that each snapshot has a hierarchy of metadata that is unique to that snapshot. This is further shown in Fig. 5 – each snapshot has its own map. It is further inherent in the copy-on-write technique that each snapshot and each active file system have unique metadata (such as block maps).
7. A method as in claim 5, wherein at least one of the snapshots is converted into a new active file system.	Siddha section 3.1.3 teaches that "[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4."

<table>
<tr>
<td></td>
<td>

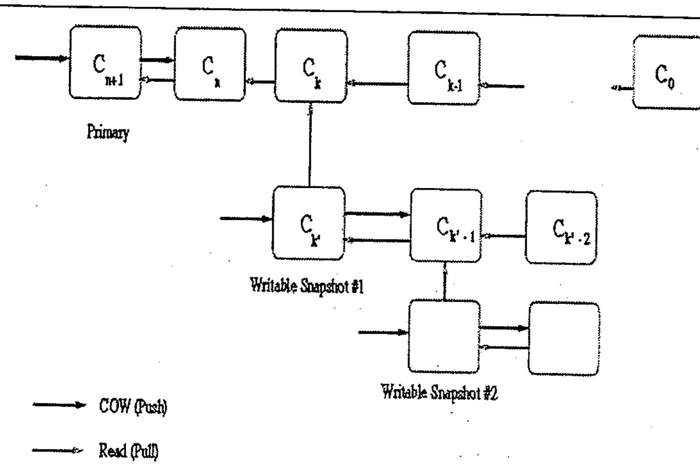


Figure 4: Snapshot Tree

</td>
</tr>
<tr>
<td>8. A method as in claim 7, wherein the one of the snapshots is converted by making the one of the snapshots writable.</td>
<td>

Siddha section 3.1.3 teaches that "[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4."

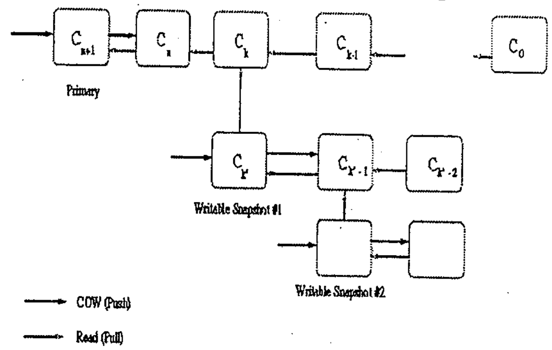
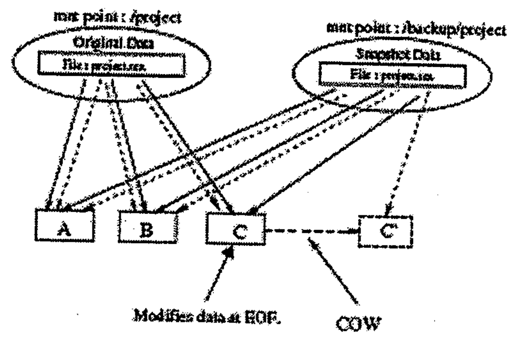
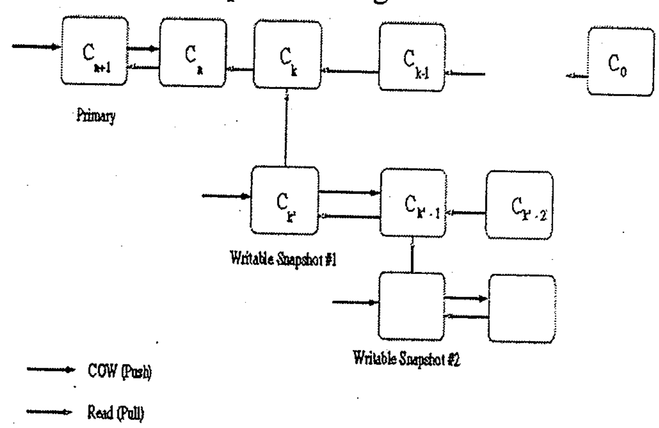


Figure 4: Snapshot Tree

</td>
</tr>
<tr>
<td>9. A method as in claim 8, wherein snapshot pointers from any of the active file systems to the new active file system are severed.</td>
<td>Siddha Fig. 4 teaches that there are no pointers between any of the active file systems. Section 3.1.3 further teaches that "[t]he basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one."</td>
</tr>
<tr>
<td>10. A method of creating plural active file systems, comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the</td>
<td>Siddha generally and section 3.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:</td>
</tr>
</table>

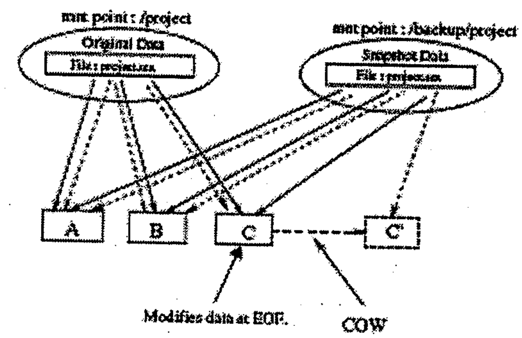
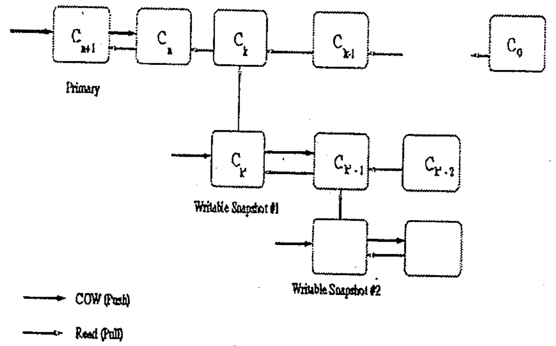
Attorney Docket No. 347155-29

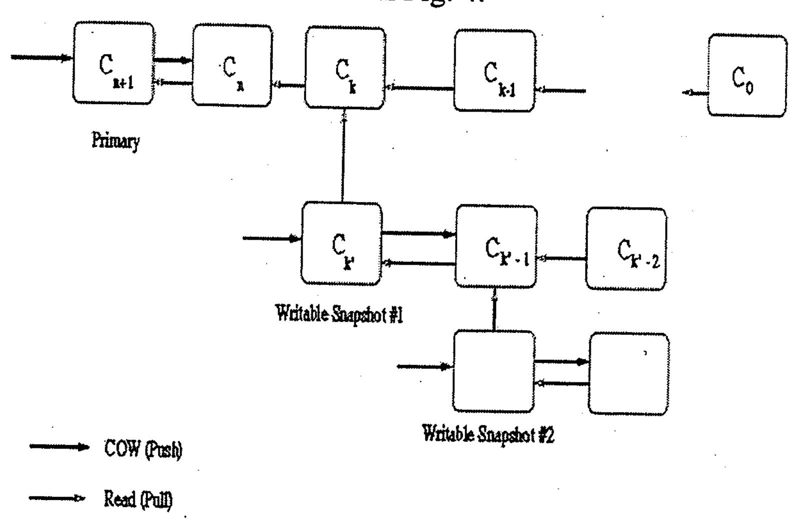
snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	 Figure 1: Snapshot COW Mechanism Siddha section 3.1.3 further teaches that snapshots can be made writable, establishing a diverging snapshot tree of multiple active file systems, as depicted in Fig. 4:  Figure 4: Snapshot Tree Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 4, writable snapshot Ck' initially shares data with snapshot Ck, which represents the state of the original file system at time k: Snapshot Ck'-1, representing the state of the new file system at a later time, does not share changed data with snapshot Ck-1, which represents the state of the original file system.
11. A method as in claim 10, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.

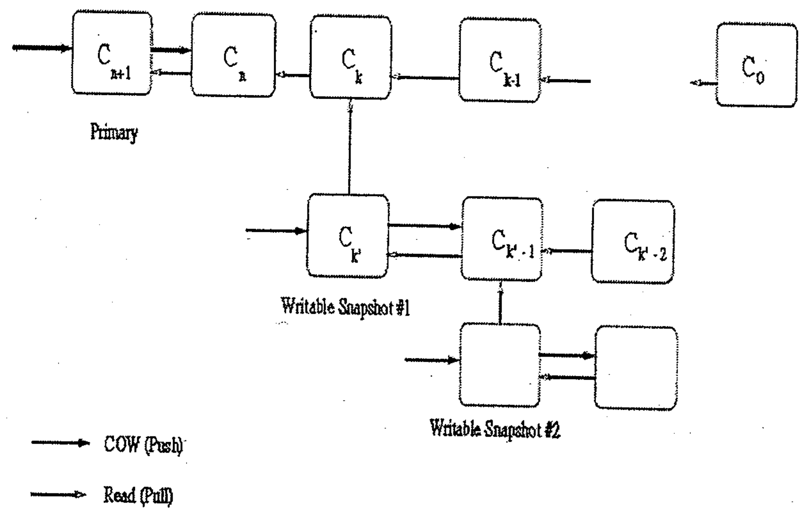
12. A method as in claim 10, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
13. A method as in claim 10, further comprising the step of severing any snapshot pointers from the first active file system to the second active file system.	Siddha Fig. 4 teaches that there are no pointers between any of the active file systems. Section 3.1.3 further teaches that "[t]he basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one."
14. A method as in claim 10, further comprising the steps of making snapshots of ones of the plural active file systems.	Siddha in section 3.1.3 and Fig. 4 teaches snapshots of plural active file systems Section 3.1.3 further teaches that "[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4.".
15. A method as in claim 14, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	Sections 1.1, 3.1.3 and Figs. 1 and 4 teach that each snapshot has a hierarchy of metadata that is unique to that snapshot. This is further shown in Fig. 5 – each snapshot has its own map. It is further inherent in the copy-on-write technique that each snapshot and each active file system have unique metadata (such as block maps).
16. A method as in claim 10, further comprising the steps of: making anew snapshot of the first active file system, the new snapshot initially sharing data with the first active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the	Siddha in section 3.1.3 and Fig. 4 teaches snapshots of plural active file systems Section 3.1.3 further teaches that "[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4." Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Changes to any of the active file systems are not reflected in the other active file systems, as shown in Fig. 4 and taught in section 3.1.3: "The basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain

third active file system.	is modified such that no downstream snapshot references this one."
17. A method as in claim 16, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
18. A method as in claim 10, further comprising the steps of: making a new snapshot of the second active file system, the new snapshot initially sharing data with the second active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	Siddha in section 3.1.3 and Fig. 4 teaches snapshots of plural active file systems Section 3.1.3 further teaches that "[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4." Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Changes to any of the active file systems are not reflected in the other active file systems, as shown in Fig. 4 and taught in section 3.1.3: "The basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one."
19. A method as in claim 18, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
20. A memory storing information including instructions, the instructions executable by a processor to operate data storage, the instructions comprising steps to maintain plural active file systems, wherein	It is inherent that the file system in Siddha is in the form of computer-executable instructions stored in a memory. Siddha generally and section 3.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:

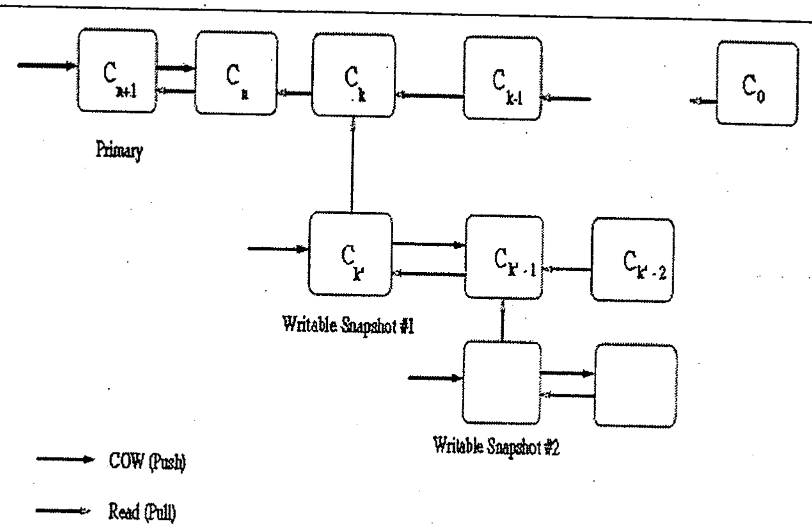
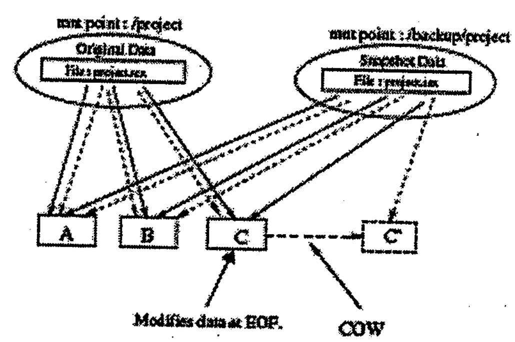
Attorney Docket No. 347155-29

each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	 Figure 1: Snapshot COW Mechanism Siddha section 3.1.3 further teaches that snapshots can be made writable, establishing a diverging snapshot tree of multiple active file systems, as depicted in Fig. 4:  Figure 4: Snapshot Tree Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 4, writable snapshot Ck' initially shares data with snapshot Ck, which represents the state of the original file system at time k. Snapshot Ck'-1, representing the state of the new file system at a later time, does not share changed data with snapshot Ck-1, which represents the state of the original file system.
21. A memory as in claim 20, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.	Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 4, writable snapshot Ck' initially shares data with snapshot Ck, which represents the state of the original file system at time k. Snapshot Ck'-1, representing the state of the new file system at a later time, does not share changed data with snapshot Ck-1, which represents the state of the original file system.
22. A memory as in claim	Due to the copy-on-write nature of snapshots disclosed in Siddha,

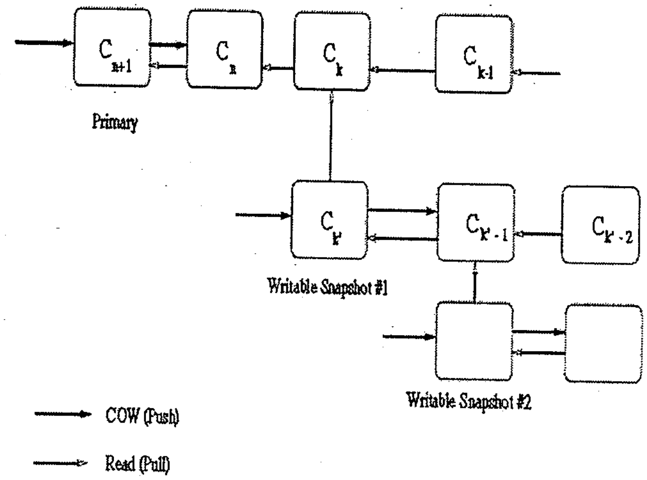
21, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
23. A memory as in claim 21, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
24. A memory as in claim 20, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.	Siddha in section 3.1.3 and Fig. 4 teaches snapshots of plural active file systems. Section 3.1.3 further teaches the need to keep snapshots consistent through atomic updates. Fig. 4 shows each snapshot to be a past consistent view of active file systems. Section 3.1.3 further teaches that "[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4."  Figure 4: Snapshot Tree
25. A method as in claim 24, wherein each snapshot includes a complete	Sections 1.1, 3.1.3 and Figs. 1 and 4 teach that each snapshot has a hierarchy of metadata that is unique to that snapshot. This is further shown in Fig. 5 – each snapshot has its own map. It is

hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	further inherent in the copy-on-write technique that each snapshot and each active file system have unique metadata (such as block maps).
26. A memory as in claim 24, wherein at least one of the snapshots is converted into a new active file system.	Siddha section 3.1.3 teaches that "[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4."  Figure 4: Snapshot Tree
27. A memory as in claim 26, wherein the one of the snapshots is converted by making the one of the snapshots writable.	Siddha section 3.1.3 teaches that "[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4."

Attorney Docket No. 347155-29

	 Figure 4: Snapshot Tree
28. A memory as in claim 27, wherein snapshot pointers from any of the active file systems to the new active file system are severed.	Siddha Fig. 4 teaches that there are no pointers between any of the active file systems. Section 3.1.3 further teaches that "[t]he basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one."
29. A memory storing information including instructions, the instructions executable by a processor to create plural active file systems, the instructions comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active	It is inherent that the file system in Siddha is in the form of computer-executable instructions stored in a memory. Siddha generally and section 3.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:  Figure 1: Snapshot COW Mechanism Siddha section 3.1.3 further teaches that snapshots can be made writable, establishing a diverging snapshot tree of multiple active file systems, as depicted in Fig. 4:

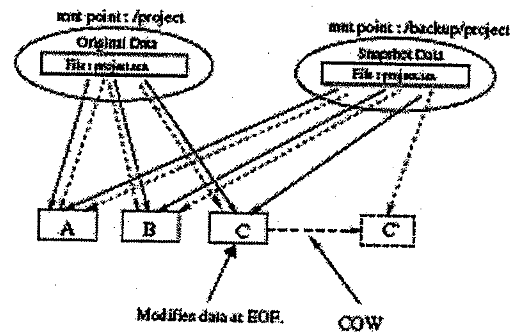
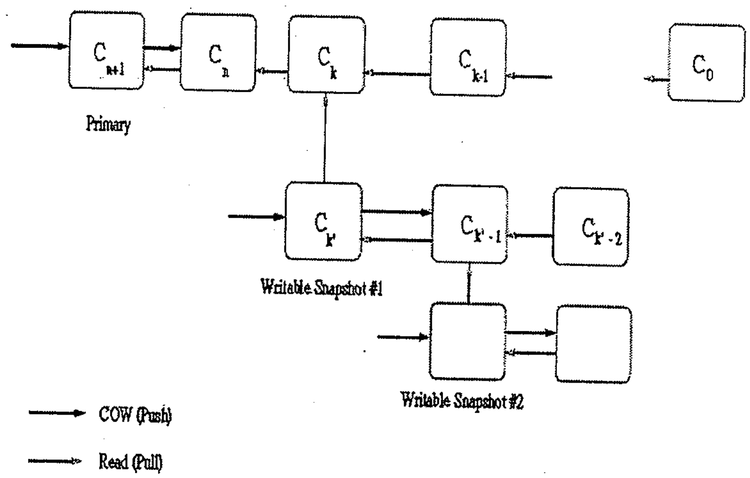
Attorney Docket No. 347155-29

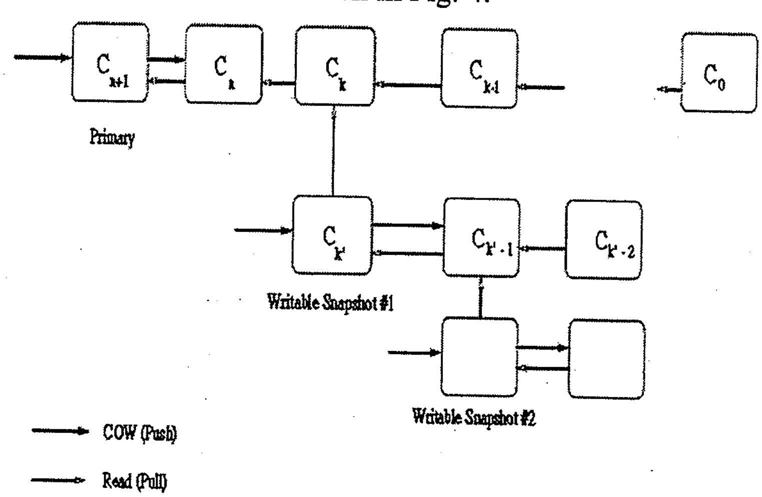
file system.	 Figure 4: Snapshot Tree Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 4, writable snapshot Ck' initially shares data with snapshot Ck, which represents the state of the original file system at time k. Snapshot Ck'-1, representing the state of the new file system at a later time, does not share changed data with snapshot Ck-1, which represents the state of the original file system.
30. A memory as in claim 29, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
31. A memory as in claim 29, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
32. A memory as in claim 29, wherein the instructions	Siddha Fig. 4 teaches that there are no pointers between any of the active file systems. Section 3.1.3 further teaches that "[t]he

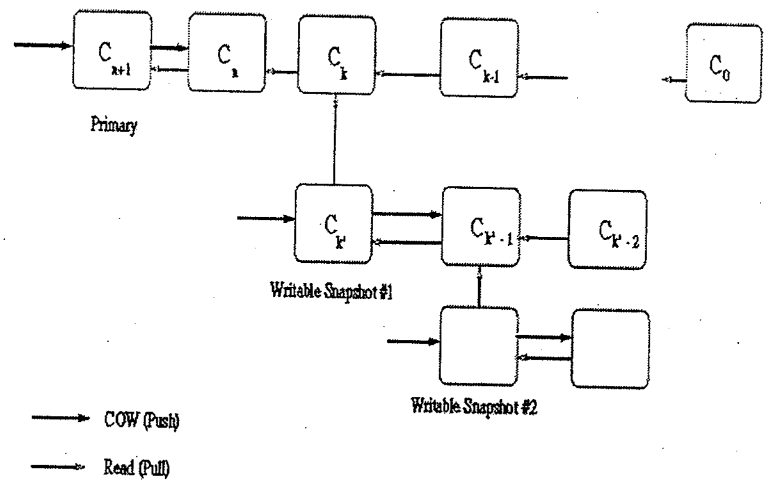
further comprise the step of severing any snapshot pointers from the first active file system to the second active file system.	basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one."
33. A memory as in claim 29, wherein the instructions further comprise the steps of making snapshots of ones of the plural active file systems.	Siddha in section 3.1.3 and Fig. 4 teaches snapshots of plural active file systems Section 3.1.3 further teaches that "[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4."
34. A memory as in claim 33, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	Sections 1.1, 3.1.3 and Figs. 1 and 4 teach that each snapshot has a hierarchy of metadata that is unique to that snapshot. This is further shown in Fig. 5 – each snapshot has its own map. It is further inherent in the copy-on-write technique that each snapshot and each active file system have unique metadata (such as block maps).
35. A memory as in claim 29, wherein the instructions further comprise the steps of: making a new snapshot of the first active file system, the new snapshot initially sharing data with the first active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	Siddha in section 3.1.3 and Fig. 4 teaches snapshots of plural active file systems Section 3.1.3 further teaches that "[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4." Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Changes to any of the active file systems are not reflected in the other active file systems, as shown in Fig. 4 and taught in section 3.1.3: "The basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one."
36. A memory as in claim 35, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the	Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.

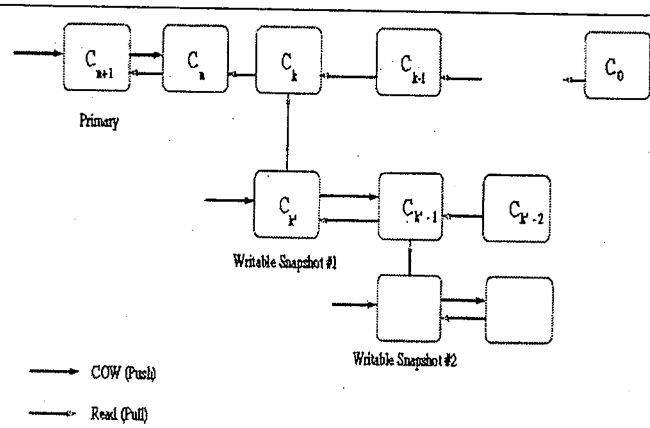
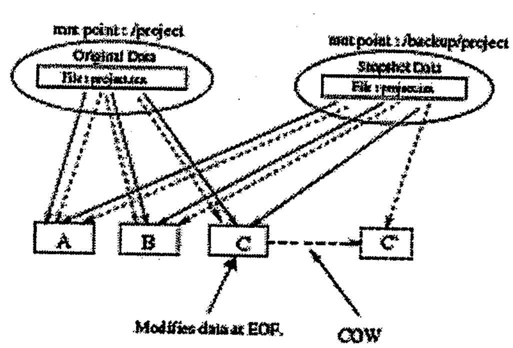
third active file system.	
37. A memory as in claim 29, wherein the instructions further comprise the steps of: making a new snapshot of the second active file system, the new snapshot initially sharing data with the second active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	Siddha in section 3.1.3 and Fig. 4 teaches snapshots of plural active file systems Section 3.1.3 further teaches that "[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4." Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Changes to any of the active file systems are not reflected in the other active file systems, as shown in Fig. 4 and taught in section 3.1.3: "The basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one."
38. A memory as in claim 37, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
39. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to maintain plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and	It is inherent that the file system in Siddha is executed in conjunction with a computer storage controller. Program and disk implementation of the Siddha file system is disclosed in section 4. One of skill in the art would readily appreciate that the Siddha file system is inherently implemented in conjunction with a computer and/or network interfaces. Siddha generally and section 3.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:

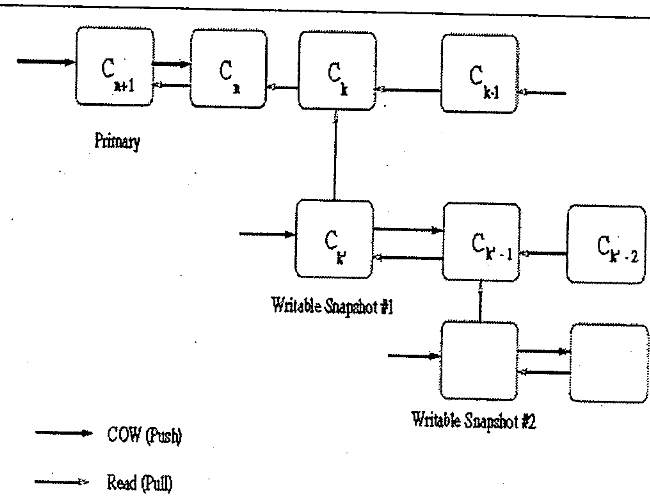
Attorney Docket No. 347155-29

wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	 Figure 1: Snapshot COW Mechanism Siddha section 3.1.3 further teaches that snapshots can be made writable, establishing a diverging snapshot tree of multiple active file systems, as depicted in Fig. 4:  Figure 4: Snapshot Tree Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 4, writable snapshot Ck' initially shares data with snapshot Ck, which represents the state of the original file system at time k. Snapshot Ck'-1, representing the state of the new file system at a later time, does not share changed data with snapshot Ck-1, which represents the state of the original file system.
40. A storage system as in claim 39, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system	Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 4, writable snapshot Ck' initially shares data with snapshot Ck, which represents the state of the original file system at time k. Snapshot Ck'-1, representing the state of the new file system at a later time, does not share changed

initially share data.	data with snapshot Ck-1, which represents the state of the original file system.
41. A storage system as in claim 40, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
42. A storage system as in claim 40, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
43. A storage system as in claim 39, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.	Siddha in section 3.1.3 and Fig. 4 teaches snapshots of plural active file systems. Section 3.1.3 further teaches the need to keep snapshots consistent through atomic updates. Fig. 4 shows each snapshot to be a past consistent view of active file systems. Section 3.1.3 further teaches that "[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4."  Figure 4: Snapshot Tree

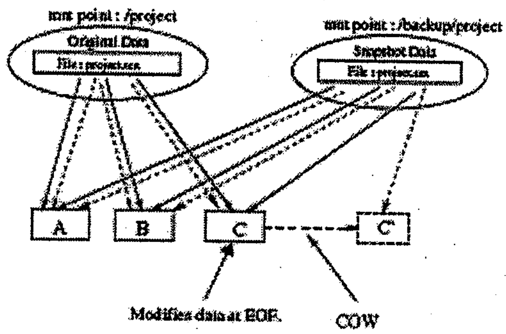
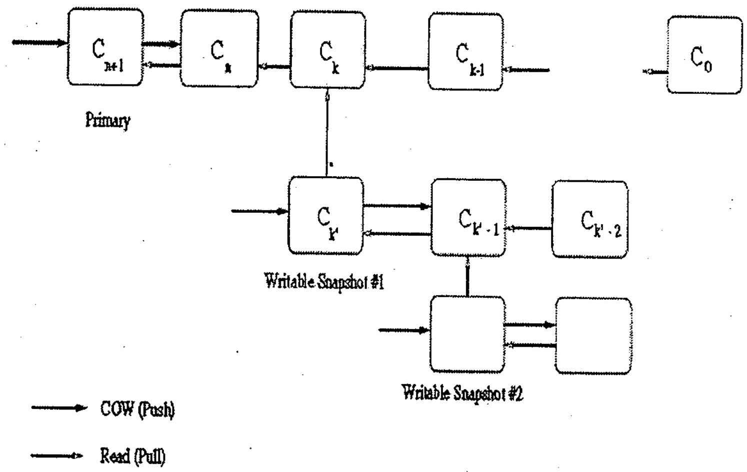
44. A storage system as in claim 43, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	Sections 1.1, 3.1.3 and Figs. 1 and 4 teach that each snapshot has a hierarchy of metadata that is unique to that snapshot. This is further shown in Fig. 5 – each snapshot has its own map. It is further inherent in the copy-on-write technique that each snapshot and each active file system have unique metadata (such as block maps).
45. A storage system as in claim 43, wherein at least one of the snapshots is converted into a new active file system.	Siddha section 3.1.3 teaches that "[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4."  Figure 4: Snapshot Tree
46. A storage system as in claim 45, wherein the one of the snapshots is converted by making the one of the snapshots writable.	Siddha section 3.1.3 teaches that "[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4."

	 Figure 4: Snapshot Tree
47. A storage system as in claim 46, wherein snapshot pointers from any of the active file systems to the new active file system are severed.	Siddha Fig. 4 teaches that there are no pointers between any of the active file systems. Section 3.1.3 further teaches that "[t]he basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one."
48. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to create plural active file systems, the program control comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not	It is inherent that the file system in Siddha is executed in conjunction with a computer storage controller. Program and disk implementation of the Siddha file system is disclosed in section 4. One of skill in the art would readily appreciate that the Siddha file system is inherently implemented in conjunction with a computer and/or network interfaces. Siddha generally and section 3.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:  Figure 1: Snapshot COW Mechanism Siddha section 3.1.3 further teaches that snapshots can be made writable, establishing a diverging snapshot tree of multiple active file systems, as depicted in Fig. 4:

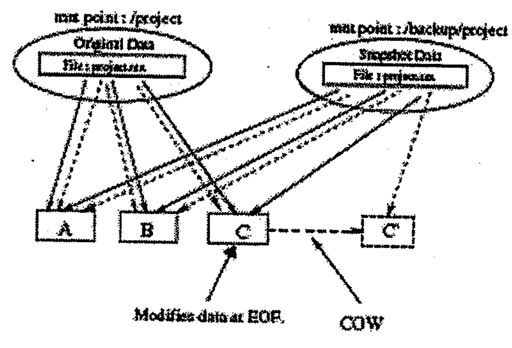
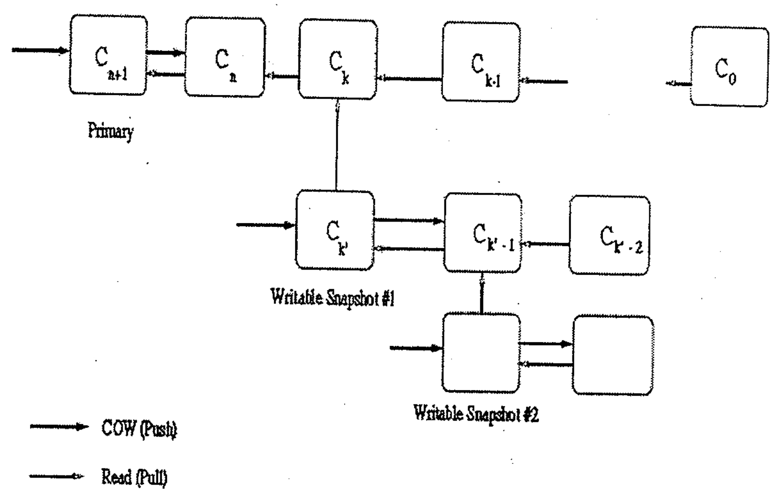
reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	 Figure 4: Snapshot Tree Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 4, writable snapshot Ck' initially shares data with snapshot Ck, which represents the state of the original file system at time k. Snapshot Ck'-1, representing the state of the new file system at a later time, does not share changed data with snapshot Ck-1, which represents the state of the original file system.
49. A storage system as in claim 48, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
50. A storage system as in claim 48, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
51. A storage system as in	Siddha Fig. 4 teaches that there are no pointers between any of

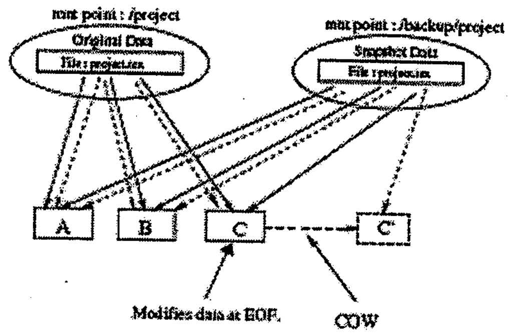
claim 48, wherein the program control further comprises the step of severing any snapshot pointers from the first active file system to the second active file system.	the active file systems. Section 3.1.3 further teaches that "[t]he basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one."
52. A storage system as in claim 48, wherein the program control further comprises the steps of making snapshots of ones of the plural active file systems.	Siddha in section 3.1.3 and Fig. 4 teaches snapshots of plural active file systems Section 3.1.3 further teaches that "[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4."
53. A storage system as in claim 52, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	Sections 1.1, 3.1.3 and Figs. 1 and 4 teach that each snapshot has a hierarchy of metadata that is unique to that snapshot. This is further shown in Fig. 5 – each snapshot has its own map. It is further inherent in the copy-on-write technique that each snapshot and each active file system have unique metadata (such as block maps).
54. A storage system as in claim 48, wherein the program control further comprises the steps of: making a new snapshot of the first active file system, the new snapshot initially sharing data with the first active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	Siddha in section 3.1.3 and Fig. 4 teaches snapshots of plural active file systems Section 3.1.3 further teaches that "[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4." Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Changes to any of the active file systems are not reflected in the other active file systems, as shown in Fig. 4 and taught in section 3.1.3: "The basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one."
55. A storage system as in claim 54, wherein when changes are made to the first	Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in

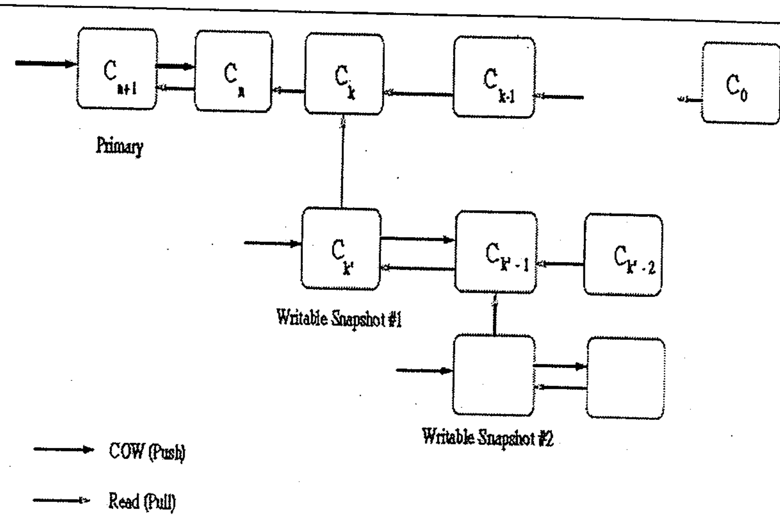
active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Fig. 1.
56. A storage system as in claim 48, wherein the program control further comprises the steps of: making a new snapshot of the second active file system, the new snapshot initially sharing data with the second active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	Siddha in section 3.1.3 and Fig. 4 teaches snapshots of plural active file systems Section 3.1.3 further teaches that "[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4." Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Changes to any of the active file systems are not reflected in the other active file systems, as shown in Fig. 4 and taught in section 3.1.3: "The basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one."
57. A storage system as in claim 56, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
58. An apparatus for operating data storage, the apparatus including means for creating plural active file systems and means for maintaining plural active file systems, wherein each of the active file systems initially access data shared with another of the active file	Siddha generally and section 3.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:

systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	 Figure 1: Snapshot COW Mechanism Siddha section 3.1.3 further teaches that snapshots can be made writable, establishing a diverging snapshot tree of multiple active file systems, as depicted in Fig. 4:  Figure 4: Snapshot Tree Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 4, writable snapshot Ck' initially shares data with snapshot Ck, which represents the state of the original file system at time k. Snapshot Ck'-1, representing the state of the new file system at a later time, does not share changed data with snapshot Ck-1, which represents the state of the original file system.
59. An apparatus for creating plural active file systems, comprising: means for making a snapshot of a first active file system, the snapshot initially sharing data with the first active file	Siddha generally and section 3.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:

Attorney Docket No. 347155-29

system; and means for converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	 Figure 1: Snapshot COW Mechanism Siddha section 3.1.3 further teaches that snapshots can be made writable, establishing a diverging snapshot tree of multiple active file systems, as depicted in Fig. 4:  Figure 4: Snapshot Tree Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 4, writable snapshot Ck' initially shares data with snapshot Ck, which represents the state of the original file system at time k. Snapshot Ck'-1, representing the state of the new file system at a later time, does not share changed data with snapshot Ck-1, which represents the state of the original file system.
60. A method of operating data storage, comprising: making a snapshot of organizational data of a first active file system, the snapshot pointing to original	Siddha generally and section 3.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:

non-organizational data of the first active file system; storing the snapshot; modifying a first portion of the original non-organizational data of the first active file system in response to a first active file system access request, resulting in a modified first portion being part of first modified non-organizational data of the first active file system; and storing the modified first portion so as not to overwrite the first portion; wherein, after the step of storing the modified first portion, the snapshot points to the original non-organizational data, the organizational data of the first active file system point to the first modified non-organizational data of the first filing system, and the original non-organizational data and the first modified non-organizational data partially overlap.	 Figure 1: Snapshot COW Mechanism As Fig. 1 shows that the snapshot originally points to data blocks A-C. When data block C is modified, the snapshot points to the original block, while the active file system points to the modified data block C'. The active file system and snapshot data sets partially overlap.
61. A method according to claim 60, wherein: the step of storing the snapshot comprises storing the snapshot as a second active filing system; the method further comprising: modifying a second portion of the original non-organizational data in response to a second active file system access request, resulting in a modified second portion being part of second modified non-	Siddha section 3.1.3 teaches that "[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4."

organizational data of the second active file system; and storing the modified second portion so as not to overwrite the second portion; wherein, after the step of storing the modified second portion, the snapshot points to the second modified non-organizational data, the organizational data of the first active file system point to the first modified non-organizational data, and the first modified non-organizational data and the second modified non-organizational data partially overlap.	 Figure 4: Snapshot Tree In Fig. 4, snapshot Ck' has been mounted as writable and originally shares data with Ck. As the two active file systems diverge over time, their data sets will partially overlap in the manner shown in Fig. 1.
62. A method according to claim 61, wherein the step of making a snapshot is performed at a consistency point of the first active file system.	Siddha sections 1.1 and 3.1.1 teach that snapshots are taken at a consistency point. Fig 4. further shows that writable snapshot Ck' is mounted at a consistency point Ck of the original active file system.
63. A method according to claim 62, wherein data is stored in the first and second active file systems using blocks.	Siddha section 1.1 and Fig. 1 teach that data is stored using blocks.

<u>Fourth Basis of Invalidity</u>

The reference applicable to the fourth basis of invalidity is:

1. Sun StorEdge Instant Image 2.0 System Administrator's Guide, February 2000 ("Sun").

The pertinence and manner of applying Sun to claims 1-59 for which re-examination is requested is as follows:

1. A method of operating data storage, the method including maintenance of plural active	Sun teaches at pp. 1-5 through 1-6 a master volume (volume containing the original data)

file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	and a shadow volume (volume that is the copy of the master volume). The shadow volume can be independent (produced via a full volume copy operation) or dependent. The dependent shadow volume is a point-in-time copy that relies on the master for all unmodified data blocks. When a data block in the master volume is modified, a copy is placed in the dependent shadow, so as to preserve point-in-time consistency of the shadow block. Thus, dependent shadow volumes use a copy-on-write technique. Sun further teaches at p. 1-6 that "[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two."
2. A method as in claim 1, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.	Sun teaches at pp. 1-5 through 1-6 that the dependent shadow volume is a point-in-time copy that relies on the master for all unmodified data blocks.
3. A method as in claim 2, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
4. A method as in claim 2, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
5. A method as in claim 1, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot.
6. A method as in claim 5, wherein each snapshot includes a complete hierarchy for file	Each shadow volume inherently includes a complete hierarchy of data that is a copy of the

system data, separate and apart from active file system data for the plural active file systems.	data on the master volume at a point in time. This includes file system data when a file system is mounted on the volumes. As the master and shadow volumes diverge over time, any changes in file system data will be unique to each volume.
7. A method as in claim 5, wherein at least one of the snapshots is converted into a new active file system.	Sun teaches at pp. 1-5 through 1-6 that once mounted, a shadow volume can be read and written to, becoming an active file system.
8. A method as in claim 7, wherein the one of the snapshots is converted by making the one of the snapshots writable.	Sun teaches at pp. 1-5 through 1-6 that once mounted, a shadow volume can be read and written to, becoming an active file system.
9. A method as in claim 8, wherein snapshot pointers from any of the active file systems to the new active file system are severed.	Because master and shadow volumes can diverge over time, they are independent and inherently do not share pointers.
10. A method of creating plural active file systems, comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	Sun teaches at pp. 1-5 through 1-6 a master volume (volume containing the original data) and a shadow volume (volume that is the copy of the master volume). The shadow volume can be independent (produced via a full volume copy operation) or dependent. The dependent shadow volume is a point-in-time copy that relies on the master for all unmodified data blocks. When a data block in the master volume is modified, a copy is placed in the dependent shadow, so as to preserve point-in-time consistency of the shadow block. Thus, dependent shadow volumes use a copy-on-write technique. Sun further teaches at p. 1-6 that "[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two."
11. A method as in claim 10, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.

12. A method as in claim 10, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
13. A method as in claim 10, further comprising the step of severing any snapshot pointers from the first active file system to the second active file system.	Because master and shadow volumes can diverge over time, they are independent and inherently do not share pointers.
14. A method as in claim 10, further comprising the steps of making snapshots of ones of the plural active file systems.	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot.
15. A method as in claim 14, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	Each shadow volume inherently includes a complete hierarchy of data that is a copy of the data on the master volume at a point in time. This includes file system data when a file system is mounted on the volumes. As the master and shadow volumes diverge over time, any changes in file system data will be unique to each volume.
16. A method as in claim 10, further comprising the steps of: making anew snapshot of the first active file system, the new snapshot initially sharing data with the first active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot. Thus, the shadow volume feature in Sun can be used to create multiple active volumes. Sun further teaches at p. 1-6 that "[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two." Thus, changes to each of the writable volumes are not reflected in the other writable volumes absent a synchronization.
17. A method as in claim 16, wherein when changes are made to the first active file system or the second active file system, modified data	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations

is recorded in a location that is not shared with the third active file system.	so as to not overwrite any other data.
18. A method as in claim 10, further comprising the steps of: making a new snapshot of the second active file system, the new snapshot initially sharing data with the second active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot. Thus, the shadow volume feature in Sun can be used to create multiple active volumes. Sun further teaches at p. 1-6 that "[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two." Thus, changes to each of the writable volumes are not reflected in the other writable volumes absent a synchronization.
19. A method as in claim 18, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
20. A memory storing information including instructions, the instructions executable by a processor to operate data storage, the instructions comprising steps to maintain plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	Sun Instant Image 2.0 software is in the form of processor-executable instructions stored in a computer memory. Sun teaches at pp. 1-5 through 1-6 a master volume (volume containing the original data) and a shadow volume (volume that is the copy of the master volume). The shadow volume can be independent (produced via a full volume copy operation) or dependent. The dependent shadow volume is a point-in-time copy that relies on the master for all unmodified data blocks. When a data block in the master volume is modified, a copy is placed in the dependent shadow, so as to preserve point-in-time consistency of the shadow block. Thus, dependent shadow volumes use a copy-on-write technique. Sun further teaches at p. 1-6 that "[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the

	shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two."
21. A memory as in claim 20, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.	Sun teaches at pp. 1-5 through 1-6 that the dependent shadow volume is a point-in-time copy that relies on the master for all unmodified data blocks.
22. A memory as in claim 21, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
23. A memory as in claim 21, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
24. A memory as in claim 20, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot.
25. A method as in claim 24, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	Each shadow volume inherently includes a complete hierarchy of data that is a copy of the data on the master volume at a point in time. This includes file system data when a file system is mounted on the volumes. As the master and shadow volumes diverge over time, any changes in file system data will be unique to each volume.
26. A memory as in claim 24, wherein at least one of the snapshots is converted into a new active file system.	Sun teaches at pp. 1-5 through 1-6 that once mounted, a shadow volume can be read and written to, becoming an active file system.
27. A memory as in claim 26, wherein the one of the snapshots is converted by making the one of the snapshots writable.	Sun teaches at pp. 1-5 through 1-6 that once mounted, a shadow volume can be read and written to, becoming an active file system.

28. A memory as in claim 27, wherein snapshot pointers from any of the active file systems to the new active file system are severed.	Because master and shadow volumes can diverge over time, they are independent and inherently do not share pointers.
29. A memory storing information including instructions, the instructions executable by a processor to create plural active file systems, the instructions comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	Sun Instant Image 2.0 software is in the form of processor-executable instructions stored in a computer memory. Sun teaches at pp. 1-5 through 1-6 a master volume (volume containing the original data) and a shadow volume (volume that is the copy of the master volume). The shadow volume can be independent (produced via a full volume copy operation) or dependent. The dependent shadow volume is a point-in-time copy that relies on the master for all unmodified data blocks. When a data block in the master volume is modified, a copy is placed in the dependent shadow, so as to preserve point-in-time consistency of the shadow block. Thus, dependent shadow volumes use a copy-on-write technique. Sun further teaches at p. 1-6 that "[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two."
30. A memory as in claim 29, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
31. A memory as in claim 29, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
32. A memory as in claim 29, wherein the instructions further comprise the step of severing any snapshot pointers from the first active file system to the second active file	Because master and shadow volumes can diverge over time, they are independent and inherently do not share pointers.

system.	
33. A memory as in claim 29, wherein the instructions further comprise the steps of making snapshots of ones of the plural active file systems.	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot.
34. A memory as in claim 33, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	Each shadow volume inherently includes a complete hierarchy of data that is a copy of the data on the master volume at a point in time. This includes file system data when a file system is mounted on the volumes. As the master and shadow volumes diverge over time, any changes in file system data will be unique to each volume.
35. A memory as in claim 29, wherein the instructions further comprise the steps of: making a new snapshot of the first active file system, the new snapshot initially sharing data with the first active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot. Thus, the shadow volume feature in Sun can be used to create multiple active volumes. Sun further teaches at p. 1-6 that "[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two." Thus, changes to each of the writable volumes are not reflected in the other writable volumes absent a synchronization.
36. A memory as in claim 35, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
37. A memory as in claim 29, wherein the instructions further comprise the steps of: making a new snapshot of the second active file system, the new snapshot initially sharing data with the second active file system; converting the new snapshot to a third active file system by making the new snapshot	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot. Thus, the shadow volume feature in Sun can be used to create multiple active volumes. Sun further teaches at p. 1-6 that "[o]nce the

writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two." Thus, changes to each of the writable volumes are not reflected in the other writable volumes absent a synchronization.
38. A memory as in claim 37, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
39. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to maintain plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	Sun Instant Image 2.0 software is in the form of processor-executable instructions stored in a computer memory. It is inherent that Sun Instant Image 2.0 software can be used in conjunction with a storage controller and a computer or network. Sun teaches at pp. 1-5 through 1-6 a master volume (volume containing the original data) and a shadow volume (volume that is the copy of the master volume). The shadow volume can be independent (produced via a full volume copy operation) or dependent. The dependent shadow volume is a point-in-time copy that relies on the master for all unmodified data blocks. When a data block in the master volume is modified, a copy is placed in the dependent shadow, so as to preserve point-in-time consistency of the shadow block. Thus, dependent shadow volumes use a copy-on-write technique. Sun further teaches at p. 1-6 that "[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two."
40. A storage system as in claim 39, wherein when a second active file system is created	Sun teaches at pp. 1-5 through 1-6 that the dependent shadow volume is a point-in-time

based on a first active file system, the first active file system and the second active file system initially share data.	copy that relies on the master for all unmodified data blocks.
41. A storage system as in claim 40, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
42. A storage system as in claim 40, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
43. A storage system as in claim 39, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot.
44. A storage system as in claim 43, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	Each shadow volume inherently includes a complete hierarchy of data that is a copy of the data on the master volume at a point in time. This includes file system data when a file system is mounted on the volumes. As the master and shadow volumes diverge over time, any changes in file system data will be unique to each volume.
45. A storage system as in claim 43, wherein at least one of the snapshots is converted into a new active file system.	Sun teaches at pp. 1-5 through 1-6 that once mounted, a shadow volume can be read and written to, becoming an active file system.
46. A storage system as in claim 45, wherein the one of the snapshots is converted by making the one of the snapshots writable.	Sun teaches at pp. 1-5 through 1-6 that once mounted, a shadow volume can be read and written to, becoming an active file system.
47. A storage system as in claim 46, wherein snapshot pointers from any of the active file systems to the new active file system are severed.	Because master and shadow volumes can diverge over time, they are independent and inherently do not share pointers.
48. A storage system, comprising: at least one storage device; an interface to at least one	Sun Instant Image 2.0 software is in the form of processor-executable instructions stored in a

computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to create plural active file systems, the program control comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	computer memory. It is inherent that Sun Instant Image 2.0 software can be used in conjunction with a storage controller and a computer or network. Sun teaches at pp. 1-5 through 1-6 a master volume (volume containing the original data) and a shadow volume (volume that is the copy of the master volume). The shadow volume can be independent (produced via a full volume copy operation) or dependent. The dependent shadow volume is a point-in-time copy that relies on the master for all unmodified data blocks. When a data block in the master volume is modified, a copy is placed in the dependent shadow, so as to preserve point-in-time consistency of the shadow block. Thus, dependent shadow volumes use a copy-on-write technique. Sun further teaches at p. 1-6 that "[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two."
49. A storage system as in claim 48, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
50. A storage system as in claim 48, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
51. A storage system as in claim 48, wherein the program control further comprises the step of severing any snapshot pointers from the first active file system to the second active file system.	Because master and shadow volumes can diverge over time, they are independent and inherently do not share pointers.
52. A storage system as in claim 48, wherein the program control further comprises the steps	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-

of making snapshots of ones of the plural active file systems.	time shadow volumes, equivalent to a snapshot.
53. A storage system as in claim 52, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	Each shadow volume inherently includes a complete hierarchy of data that is a copy of the data on the master volume at a point in time. This includes file system data when a file system is mounted on the volumes. As the master and shadow volumes diverge over time, any changes in file system data will be unique to each volume.
54. A storage system as in claim 48, wherein the program control further comprises the steps of: making a new snapshot of the first active file system, the new snapshot initially sharing data with the first active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot. Thus, the shadow volume feature in Sun can be used to create multiple active volumes. Sun further teaches at p. 1-6 that "[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two." Thus, changes to each of the writable volumes are not reflected in the other writable volumes absent a synchronization.
55. A storage system as in claim 54, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
56. A storage system as in claim 48, wherein the program control further comprises the steps of: making a new snapshot of the second active file system, the new snapshot initially sharing data with the second active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot. Thus, the shadow volume feature in Sun can be used to create multiple active volumes. Sun further teaches at p. 1-6 that "[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A

	bitmap volume file tracks the differences between the two." Thus, changes to each of the writable volumes are not reflected in the other writable volumes absent a synchronization.
57. A storage system as in claim 56, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
58. An apparatus for operating data storage, the apparatus including means for creating plural active file systems and means for maintaining plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	Sun teaches at pp. 1-5 through 1-6 a master volume (volume containing the original data) and a shadow volume (volume that is the copy of the master volume). The shadow volume can be independent (produced via a full volume copy operation) or dependent. The dependent shadow volume is a point-in-time copy that relies on the master for all unmodified data blocks. When a data block in the master volume is modified, a copy is placed in the dependent shadow, so as to preserve point-in-time consistency of the shadow block. Thus, dependent shadow volumes use a copy-on-write technique. Sun further teaches at p. 1-6 that "[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two."
59. An apparatus for creating plural active file systems, comprising: means for making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and means for converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	Sun teaches at pp. 1-5 through 1-6 a master volume (volume containing the original data) and a shadow volume (volume that is the copy of the master volume). The shadow volume can be independent (produced via a full volume copy operation) or dependent. The dependent shadow volume is a point-in-time copy that relies on the master for all unmodified data blocks. When a data block in the master volume is modified, a copy is placed in the dependent shadow, so as to preserve point-in-time consistency of the

	shadow block. Thus, dependent shadow volumes use a copy-on-write technique. Sun further teaches at p. 1-6 that "[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two."

Other References

1. C. Czezatke, M. Anton Ertl, *LinLogFS – A log-structured Filesystem For Linux*, Proceedings of FREENIX Track: 2000 USENIX Annual Technical Conference, June 18-23, 2000 (hereinafter: "Czezatke").

Czezatke discloses writable snapshots of a file system, thereby anticipating or rendering obvious the claims of the '001 patent. Figs. 1 and 2 disclose copy-on-write techniques for forming file system snapshots and further disclose usage of data blocks and file system metadata, such as superblocks and inodes. Section 2.2 further explains the copy-on-write methodology. Writable snapshots are disclosed in the Introduction and further disclosed in section 4 as a direction for future work.

2. The Enterprise Challenge Served By Snapshot, LSI Logic Whitepaper, 2001 (hereinafter: "LSI Logic Whitepaper").

LSI Logic Whitepaper discloses writable snapshots of a file system, thereby anticipating or rendering obvious the claims of the '001 patent. Copy-on-write snapshots are disclosed at pp. 3-4. Page 1 discloses that the snapshots can be "read, written and copied." Page 4 further teaches that "[t]he SANtricity snapshot feature is able to read, write and copy the snapshot. The writes, or updates to the snapshot, are handled in the repository. If a write occurs to the snapshot, it overwrites the PiT image with the change, however, this is a valuable capability that opens up new techniques for creating immediate test and small data mining environments. This is also a distinguishing capability of LSI Logic's snapshot feature." Thus, the LSI Logic Whitepaper discloses that writable snapshots diverge from the parent file system as the modified data is not shared between writable snapshots and parent file systems.

3. N. Osorio and B. Lee, *Guidelines for Using Snapshot Storage Systems for Oracle Databases*, version 1 dated August 28, 2000 (hereinafter: "Osorio").

Osorio discloses file systems using copy-on-write snapshots that can be written to, thereby anticipating or rendering obvious the claims of the '001 patent. Pp. 3-4 disclose copy-on-write snapshots as a "copy image of storage devices or file systems." P. 7 further teaches that the snapshot can be converted into another active file system that initially shares data with the original system, but diverges over time: "When a second host can read and write the snapshot image, the second host can start a second Oracle instance. . . . For this usage, the snapshot area essentially is a new database started with a copy of the original database."

4. U.S. Pat. 6,341,341 to Grummon et al. (hereinafter "Grummon").

Grummon discloses a file system using copy-on-write snapshots and having a writable snapshot, thereby anticipating or rendering obvious the claims of the '001 patent. Fig. 3 discloses a file system 300. Read-Write On-Line Container 310 represents the active file system while Read-Write Snapshot Container 308 represents the writable snapshot. Col. 7: 11-16. The snapshot and active file system initially share data. *See e.g.* Col. 6: 37-47. As data is written to container 308, changes are not shared. *See e.g.* Col. 7:17-63.

5. U.S. Pat. 5,675,802 to Allen et al. (hereinafter "Allen").

Allen teaches a file system for viewing selected versions of the files stored in a versioned object base (VOB). Col. 6:16-18, Fig. 2. A virtual file system is provided that allows users to have a private view of the files. Col. 6:49-58. Allen further teaches "branching" of the file trees, with each of the branches undergoing independent evolution from parent and sister branches. Col. 6:59-7:223, Fig. 3. As shown in Fig. 3, a branch initially shares data with the parent, but diverges over time, with changes not shared with the other branches. Therefore, Allen anticipates or renders obvious the claims of the '001 patent.

6. U.S. Pat. 6,795,966 to Lim et al. (hereinafter: "Lim").

Lim teaches a computer system state checkpoint mechanism. Multiple checkpoints can be taken. *See e.g.* Col. 6:45-65, 8:5-7. The checkpoints can be mounted on another virtual machine and used as an initial state, diverging over time as the virtual machine operates. *See e.g.* Col. 7: 6-18, 8:8-14. Therefore, Lim anticipates or renders obvious the claims of the '001 patent.

II. STATEMENT POINTING OUT SUBSTANTIAL NEW QUESTION OF PATENTABILITY

Since claims 1-63 of the '001 Patent are not patentable over the prior art references cited above for the reasons set forth above, a substantial new question of patentability is raised for each claim. Further, these prior art references cited above are material to the subject matter of the '001 Patent. In particular, these prior art references provide teachings not provided during the prosecution of the '001 Patent. Therefore, a substantial new question of patentability has been raised, and reexamination is respectfully requested.

CONCLUSION

Based on the above remarks, it is respectfully submitted that a substantial new question of patentability has been raised with respect to Claims 1-63 of the '001 Patent. Therefore, reexamination of Claims 1-63 is respectfully requested.

Any fee due for this reexamination may be charged to Deposit Account No. 07-1896.

Respectfully submitted,

DLA PIPER US LLP

Date: November 30, 2007

By: Ronald L. Yin
Ronald L. Yin
Reg. No. 27,607

Attorneys for Applicant(s)

Ronald L. Yin
DLA Piper US LLP
2000 University Avenue
East Palo Alto, CA 94303-2248
650-833-2437 (Direct)
650-833-2000 (Main)
650-833-2001 (Facsimile)
ronald.yin@dlapiper.com

Substitute for form 1449A/PTO

INFORMATION DISCLOSURE STATEMENT BY APPLICANT

(Use as many sheets as necessary)

Sheet 1 of 2

Complete if Known	
Patent Number	6,857,001
Issue Date	February 15, 2005
First Named Inventor	Hitz et al.
Art Unit	N/A
Examiner Name	N/A
Attorney Docket Number	347155-29

U. S. PATENT DOCUMENTS

Examiner Initials*	Cite No.[1]	Document Number Number-Kind Code[2] (if known)	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
		US- 6,341,341	01-22-2002	Grummon et al.	
		US- 5,675,802	10-07-1997	Allen et al.	
		US- 6,795,966	09-21-2004	Lim et al.	
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			
		US-			

FOREIGN PATENT DOCUMENTS

Examiner Initials*	Cite No.[1]	Foreign Patent Document Country Code[3] "Number[4] "Kind Code[5] *(if known)*	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear	T[6]

Examiner Signature		Date Considered	

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant. [1]Applicant's unique citation designation number (optional). [2]See Kinds Codes of USPTO Patent Documents at www.uspto.gov or MPEP 901.04. [3]Enter Office that issued the document, by the two-letter code (WIPO Standard ST.3). [4]For Japanese patent documents, the indication of the year of the reign of the Emperor must precede the serial number of the patent document. [5]Kind of document by the appropriate symbols as indicated on the document under WIPO Standard ST. 16 if possible. [6]Applicant is to place a check mark here if English language Translation is attached.

This collection of information is required by 37 CFR 1.97 and 1.98. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 2 hours to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: **Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

Substitute for form 1449B/PTO

INFORMATION DISCLOSURE STATEMENT BY APPLICANT

(Use as many sheets as necessary)

Sheet 2 of 2

Complete if Known	
Patent Number	6,857,001
Issue Date	February 15, 2005
First Named Inventor	Hitz et al.
Art Unit	N/A
Examiner Name	N/A
Attorney Docket Number	347155-29

NON PATENT LITERATURE DOCUMENTS

Examiner Initials*	Cite No.[1]	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc.), date, page(s), volume-issue number(s), publisher, city and/or country where published.	T[2]
		Hitz et. al, *File System Design For An NFS File Server Appliance*, TR3002, USENIX January 19, 1994 (hereinafter: "Hitz").	
		Ylonen et. al, *Concurrent Shadow Paging: Snapshots, Read-Only Transactions, and On-The-Fly Multilevel Incremental Dumping*, TKO-B104, Laboratory of Information Processing Science at Helsinki University of Technology, 1993 (hereinafter: "Ylonen"). Although this document is undated on its face, it is referenced in Ylonen et. al, *Concurrent Shadow Paging: Fine-Granularity Locking with Support for Extended Lock Modes and Early Releasing of Locks*, Laboratory of Information Processing Science at Helsinki University of Technology (see Footnote [17], page 28), as being published in 1993.	
		Veritas File System 3.4 Administrator's Guide, November 2000 ("VxFS").	
		S. B. Siddha, K. Gopinath, *A Persistent Snapshot Device Driver for Linux*, Proceedings of the 5th Annual Linux Showcase & Conference, USENIX, Nov. 5-10, 2001 (hereinafter: "Siddha").	
		S. B. Siddha, *Persistent Snapshots*, A Project Report Submitted in partial fulfillment of the requirement for the Degree of Master of Engineering in Computer Science and Engineering, Indian Institute of Science, January, 2000 (hereinafter, "Siddha Report").	
		Sun StorEdge Instant Image 2.0 System Administrator's Guide, February 2000 (hereinafter: "Sun").	
		C. Czezatke, M. Anton Ertl, *LinLogFS – A log-structured Filesystem For Linux*, Proceedings of FREENIX Track: 2000 USENIX Annual Technical Conference, June 18-23, 2000 (hereinafter: "Czezatke").	
		The Enterprise Challenge Served By Snapshot, LSI Logic Whitepaper, 2001 (hereinafter: "LSI Logic Whitepaper").	
		N. Osorio and B. Lee, *Guidelines for Using Snapshot Storage Systems for Oracle Databases*, version 1 dated August 28, 2000 (hereinafter: "Osorio").	

Examiner Signature		Date Considered	

* EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.
[1] Applicant's unique citation designation number (optional). [2] Applicant is to place a check mark here if English language Translation is attached.